Unix MSNL Code

Richard Black

7th May 1993

1 Introduction

This document describes the interface for the second generation unix MSNL implementation which uses kernel data paths and user space control.

The interface consists of messages passed over a back door raw socket. The interface is liable to change from the one described herein. The interface is unashamedly optimised for the kernel for implementation reasons.

There are three different types of message which are sent between the kernel and user space code. These deal with the management of sockets, the management of associations and the initialisation of the port code.

2 Management Messages

Management messages have the following fields. Not all of these fields are disjoint. Some are overlapping and have meaning only for certain messages. The fields are as described in table 1.

Type	Name
int	mm_type
int	mm_request
int	mm_reply
void *	mm_manage
struct sockaddr	mm_addr
unsigned char [6]	mm_src
int	mm_associd
int	mm_pod
void *	mm_pcb
int	mm_error

Table 1: Management Message Fields

The fields mm_manage and mm_pcb are opaque pointer objects which may not be dereferenced in user space.

The field mm_type specified which type of operation is being performed. It is one of MM-TYPE_SOCKET, MMTYPE_ASSOC or MMTYPE_PORT. These three different types will now be discussed.

3 Socket Management

This is the largest (and most finished) section of the interface. There are six different messages sent up from the kernel and seven sent down to the kernel. Messages up have a command in the mm_request field. Messages down have a command in the mm_reply field.

A note on errors which "Shouldn't happen." This is shouldn't happen if the user space manager is doing its job correctly. It is reported when a downwards command is received for a protocol control block which is waiting for a MMDOWN_FINAL when the command is not a MMDOWN_FINAL.

3.1 MMUP_BIND

This message informs the user space about a new socket and the address that has been assigned to it. Unfortunately the unix socket implementation is such that the bind code has to be done in the kernel. This is therefore a notification rather than a request. The fields are:

mm_manage This contains the opaque allocated by the kernel for this new socket.

mm_addr This contains the address bound.

3.2 MMUP_LISTEN

This message indicates that a listen has been performed on a socket. The kernel only permits this in the bound state. The fields are:

mm_manage This contains the opaque which identifies the socket.

3.3 MMUP_CONNECT

This message indicates that a request has been made to connect on a given socket. The kernel only permits this in the bound state. The socket is set to the connecting state. The fields are:

mm_manage This contains the opaque which identifies the socket.

mm_addr This contains the address which the user wishes to connect to.

3.4 MMUP_ACCEPT

This message is sent when a user accept(2)'s a socket which is in state acceptable. The socket goes into the accepting state and the unix code believes the socket is connected. Data sent will now be binned. The fields are:

mm_manage This contains the opaque which identifies the socket. Note that this opaque will be new to the user space manager because this is a child socket.

mm_addr The destination (peer) address that the user has picked up for this socket. This information allows the user manager to know what connection is involved.

3.5 MMUP_DISCONNECT

This message is sent when a socket is disconnected. The manager should tear down the connection. The socket is now in the disconnected state. The data path shutdown routine will be called. It will be issued for both sides of a local connection and for a socket which is explicitly dicsonnected with MMDOWN_DISCONNECT. This can occur once the unix socket code believes the socket is connected, whether it is or not.

mm_manage	This contains the opaque which identifies the socket.

3.6 MMUP_DETACH

This message is sent when a socket is detached or aborted. The manager may need to tear down the connection if not already disconnected. The socket is now in the dead state awaiting destruction acknowledgement.

mm_manage	This contains the opaque which identifies the socket.

3.7 MMUP_CONNECTTIMEDOUT

This message is sent when a connect on a socket is timed out by the kernel. The manager will need to cease attempting to connect the socket.

mm_manage	This contains the opaque which identifies the socket.

3.8 MMDOWN_NEWCONN

This message is sent down from the user to indicate that a listening socket has had a new connection come in. The kernel creates a new daughter socket which goes into state acceptable. The fields are:

mm_manage	This contains the opaque which identifies the master (listening) socket.
mm_addr	This indicates the peer address.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EINVAL	The socket is not in the listening state.
ENOBUFS	Kernel memory exhausted.

This call should always be issued as an ioctl. This is to permit the new opaque reference for the new socket to be returned to the manager. This is returned in the mm_manage field.

3.9 MMDOWN_ACCEPTED

This message is sent down following a MMUP_ACCEPT to indicate that the data paths are now established. The socket goes into the connected state and data can then be transmitted. This is not used for local connections. The fields are:

mm_manage	This contains the opaque which identifies the socket.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EINVAL	The socket is not in the accepting state.

3.10 MMDOWN_CONNECTED

This message informs an active connector that the connection has been established. The socket goes into the connected state. This is not used for local connections. The fields are:

mm_manage	This contains the opaque which identifies the socket.
mm_addr	This contains the address of the peer.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EINVAL	The socket is not in the connecting state.

3.11 MMDOWN_DISCONNECT

This message allows the user manager to disconnect a socket. This must be performed before removing the data paths. The unix code is informed that the socket is disconnected. The data path shutdown routine will be called. This can be done once the unix socket code believes the socket is connected, whether it is or not. The fields are:

mm_manage	This contains the opaque which identifies the socket.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.

3.12 MMDOWN_ABORT

This message allows the user manager to abort (nuke) a socket. This is highly destructive. This translates to the unix soabort call. The fields are:

mm_manage	This contains the opaque which identifies the socket.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.

3.13 MMDOWN_FINAL

This message is the acknowledgement that the user manager has relinquished all the state associated with the opaque object and will never issue it again. The socket must be in state dead. All memory associated with the socket is released. The fields are:

mm_manage	This contains the opaque which identifies the socket.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EBUSY	The socket is not dead!

3.14 MMDOWN_SOCKETJOIN

This message is used to establish a local MSNL connection. It is a special case. It must be issued on the connecting socket. The other socket must be in state accepting. Both sockets become connected. This call replaces MMDOWN_CONNECTED and MMDOWN_ACCEPTED for local connections. The fields are:

mm_manage	This contains the opaque for the connecting socket.
mm_pcb	This contains the opaque for the accepting socket.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	Either opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EINVAL	The sockets are not in the correct state.

3.15 MMDOWN_CONNECTFAIL

This message informs an active connector that the connection cannot be established. The reason is specified. The socket goes into the disconnected state. The fields are:

mm_manage	This contains the opaque which identifies the socket.
mm_error	This number will be passed to the connecting client in *errno*.

The possible errors generated by the kernel MSNL code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.
EFAULT	The opaque does not specify a valid pcb.
ENOTSOCK	Shouldn't happen.
EINVAL	The socket is not in the connecting state.

4 Association Management

This section has three downwards commands and three upwards ones.

4.1 MMUP_EDL

This indicates that a promiscuous ethernet packet has arrived. The EDL packet *data* follows the manage_msg structure. The fields are:

mm_src	The first six octets contain the ethernet address of the sender. This is unfortunatley required for the current version of EDL Meta-Signalling.
mm_pod	This contains the pod that the EDL Metasignalling was received on.

4.2 MMUP_ASSOCBAD

This indicates that a packet has been received on an association which is not currently valid. The manager should perform whatever signalling is required to quench the source. The manager should regard this as an expedited operation. The fields are:

mm_associd	The offending association identifier.
mm_pod	The pod on which it came in.

4.3 MMUP_FDL

This indicates that a meta signalling FDL cell has arrived. The FDL cell follows the manage_msg structure. The fields are:

mm_pod	This contains the pod that the FDL Metasignalling was received on.

4.4 MMDOWN_ASSOCCREATE

This is used to create the association data path for a socket. It gives all the information necessary to prepare tables in the drivers etc. No assoc destroy is provided because the destruction call-backs are called automatically when the socket is disconnected, detached or aborted. The valid fields are:

mm_pcb	This contains the opaque reference to the socket for which this is the data connection being created.
mm_pod	This specifies the interface number on which the association is being made.
mm_associd	The association identifier.
mm_addr	This contains the transmission vci appropriate for the association and the interface. For the ethernet this is eight octets sa_data[0] through sa_data[7]. For FDL this is four octets including two routing bytes – for the yes board this is two octets in sa_data[2] and sa_data[3].

The possible errors generated by the kernel association code are:

EMSGSIZE	The message is not large enough to contain a struct manage_msg.

EFAULT — The mm_pcb field does not specify a valid pcb.

EINVAL — The specified socket already has a data path, or the association number is out of range.

ENXIO — The specified pod does not exist.

EEXIST — The association identifier is already in use.

ENOBUFS — Kernel memory exhausted.

4.5 MMDOWN_ASSOCSEND

This is used to send a meta-signalling packet on the specified pod. The code for this isn't 100% finished yet. The data to be sent follows the struct manage_msg. The valid fields are:

mm_pod — This specifies the interface number on which the meta signalling is to be sent.

mm_addr — This contains the transmission vci appropriate for the interface. For the ethernet this is eight octets sa_data[0] through sa_data[7]. For FDL this is four octets including two routing bytes – for the yes board this is two octets in sa_data[2] and sa_data[3].

The possible errors generated by the kernel association code are:

EMSGSIZE — The message is not large enough to contain a struct manage_msg and some data.

ENXIO — The specified pod does not exist.

ENOBUFS — Kernel memory exhausted.

4.6 MMDOWN_ASSOCIDLE

This is used to allow the manager to inquire if an association has had any traffic on it, and is for the purposes of automatic tearing down of unused circuits. This call will mark the association as idle and return the current value of its idle flag. Assocations are created in the idle state. This call should clearly only be used in the ioctl form. Fields used are:

mm_associd — This is the association identifier to be queried

mm_reply — This is *updated* to the current value of the idle flag.

5 Port management

This section contains a single command for initialising the bind code which unfortunately has to be in the kernel.

5.1 Setting the host address

The bind code must set the users msnl address for subsequent sockaddr(2) calls. This sets the address that will be given. It is used for no other purpose. It may be changed at any time and does not retrospectively modify the sockets already in existance. The command is MMDOWN_HOSTSET and the fields are:

mm_addr — This is cast into a struct sockaddr_msnl and the machines host address is set to the m_host field.

The possible errors generated by the kernel port code are:

EMSGSIZE — The message is not large enough to contain a struct manage_msg.

6 IP Tunnelling code

The unix kernel code optionally contains four logical interfaces called *ms0* through *ms3* which uses the MSNL code to implement an IP over MSNL tunnelling service. The tunnelling code uses MSNL connections controlled by the user space manager in a way which is very similar but not identical to user space MSNL sockets. The differences are to do with the interconversion of IP and MSNL addresses. The only difference between the four interfaces is that they use a different MTU value. IP routeing should be used to configure routeing for remote machines as required.

6.1 Binding

When the tunnelling code interface is given an address with *ifconfig* it will bind and listen on an MSNL socket for incoming connections. The address it supplies to the bind must be an AF_MSNL address, and it leaves the MSNL address field zero, and requests one of the four IP tunnelling ports (depending on which interface it is).[1] Thus it will listen on the machine's default MSNL address. The IP address which is being used on that interface is communicated to the user space manager in the *ms_ipa* field in the address.

6.2 Connecting

Since the tunnelling code must leave the mapping of IP to MSNL addresses to the management code the tunnel passes an AF_INET style address to the connect. The manager must perform the translation and make the connection to the appropriate place. However the code must also communicate the MTU that it is going to use since the connection must be to the MSNL tunneling interface on the remote machine which uses the same MTU value. To help the manager perform the correct connection, the IP tunneling code places the port number of the appropriate IP MTU / MSNL service in the four octets sa_data[6] through sa_data[9] of the connection address.

6.3 Accepting

For the purposes of ensuring the end to end existence and validity of a tunnel, tunnels are used in a full-duplex manner. As a result the tunnelling code must always be informed of the address of the IP peer when a new connection to it is established. The user space manager *must*, therefore, present an AF_INET style address when issuing an MMDOWN_NEWCONN.

[1] See the assigned numbers document